VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Návrh a implementace herního jádra a hry pro mobilní telefon

Design and Implementation of a Game Engine and a Game for a Mobile Phone

2010

Martin Lipovský

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Rád bych na tomto místě poděkoval všem, kteří mi pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací obecného herního jádra a jednodušší hry pro mobilní telefon s operačním systémem Android OS. Toto jádro bude implementováno v programovacím jazyce Android, který vychází z Javy.

Vytvořená hra bude pro jednoho hráče. Hra je ve stylu 'shoot-em-up' s jednoduchou pseudo 3D grafikou. Jednotlivé levely můžeme editovat na PC. Pro použití herního jádra a nainstalování hry musíme mít na cílovém zařízení nainstalován Android minimálně verze 1.6.

Klíčová slova: Android, scrolling shooter, emulátor, Dalvik VM

Abstract

This thesis deals with the design and implementation of the general game core and simple game for mobile device with operation system Android OS. This core is implemented in programming languages Android which is based on Java.

The game is for one player. The game is in the style of 'shoot-em-ups' with a simple pseudo-3D graphics. Game levels can edit on your PC. To use the gaming kernel and install the game must be installed Android at least version 1.6 on the target device.

Keywords: Android, scrolling shooter, emulator, Dalvik VM

Seznam použitých zkratek a symbolů

AVD	–	Android Virtual Devices
DDMS	–	Dalvik Debug Monitor Service
VM	–	Virtual Machine

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Mobilní telefony jsou v dnešní době již běžnou součástí našeho života. Už ale dávno neslouží pouze ke komunikaci s ostatními, ale nabízejí spoustu dalších funkcí, které nám mohou zpříjemňovat život při běžných denních činnostech. Hlavně mezi mladými uživateli, jsou mobilní telefony s oblibou využívány k zábavě a hraní různých mobilních her. Mobilní hry ale nejsou pouze hry určené pro mobilní telefony, patří sem i hry hrané na smartphonech, PDA, kapesních počítačích a přenosných hudebních přehrávačích.

První hry v mobilech se začaly objevovat v roce 1997 s příchodem Nokie 6110 a hlavně legendární Nokie 5110, která nabízela plně grafickou zobrazovací jednotku a 3 vestavěné hry. Velkou předností těchto her byla jejich hratelnost, ovšem grafika vzhledem k monochromatickému displeji byla strohá. Většina mobilů s tímto displejem měla pouze předinstalované hry a přidat nějakou další hru bylo velmi obtížné, ne-li zcela nemožné. Telefony této doby používaly rozdílné systémy a také měli velmi omezené systémové prostředky. Postupem času se začal zvyšovat výkon a paměť mobilních telefonů, dalším posunem byl barevný displej, který ve svých počátcích byl schopen zobrazovat 256 barev.

Asi největší zlom ve vývoji her pro mobilní telefony nastal v roce 2000, kdy Sun Microsystem vytvořil standard zvaný MIDP pro uvedení Sun Java software pro mobilní zařízení.

Tímto krokem nabral vývoj her úplně jiný směr, protože nebylo třeba předělávat hru pro různé druhy telefonů. Tento standard začaly používat největší světoví výrobci mobilů a používá se dodnes.

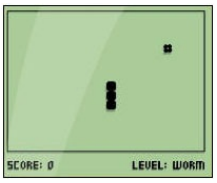
V současnosti jsou již mobily a mobilní zařízení natolik výkonné, že bez problému zvládají různé operační systémy. Mezi nejznámější patří Symbian, používaný hlavně u Nokií, dále Windows Mobile, iPhone OS a stále ještě nováček Google Android.

Cílem této bakalářské práce je vytvořit herní jádro a hru postavenou na tomto jádře. Pro vytvoření této hry jsem si vybral platformu Android firmy Google. Hra by měla být typu shoot-em-up. Tato hra bude určena pro jednoho hráče s jednoduchou grafikou. [1]

2 Historie a možnosti vývoje her pro mobilní telefony

2.1 Počátek mobilních her

Historie mobilních her se začala psát s vydáním, dnes asi nejslavnější, mobilní hry „Snake“. Jednalo se o vedení jakéhosi „hada“, představovaného tenkou čarou, po obrazovce a sbírání bodů.



Obrázek 1: Ukázka hry Snake [1]

2.2 Uvedení WAPu

Dalším milníkem ve vývoji mobilních her bylo uvedení WAPu a s ním tzv. WAP her. WAP je technologický standard vyvinutý pro umožnění spojení na internet pomocí mobilního telefonu. Nejvíce se na vývoji podílela firma Unwired Planet, společně s dalšími výrobci mobilních telefonů spustila WAP Forum a zajistila standardizaci v Evropě. Výsledkem jejich snažení bylo vytvoření malého internetového prohlížeče pro telefony, který umožňoval připojení k serveru a přenos dat do telefonu. Tvůrci her byl WAP využíván jako spojovací technologie pro jednoduché ovládání, multiplayerových, zvláště deskových her, kde nebyla vyžadována velká rychlost spojení. Hlavní přínos WAPu pro tvůrce mobilních her, byl ale v možnosti tvořit hry a poskytovat je zákazníkům prostřednictvím sítě. Tímto počinem odstartoval BOOM mobilních her mezi uživateli. Ti už nebyli limitování pouze předinstalovanými hrami, ale mohli si stáhnout jakoukoli další hru, kterou jejich telefon podporoval.

2.3 Příchod Javy

V roce 2000 bylo největší událostí zavedení programovacího prostředí Java, oficiálně nazvaného Java 2 Micro Edition resp. J2ME, později Java ME. Jedná se o platformu určenou pro mobilní zařízení a zabudované systémy. [5]

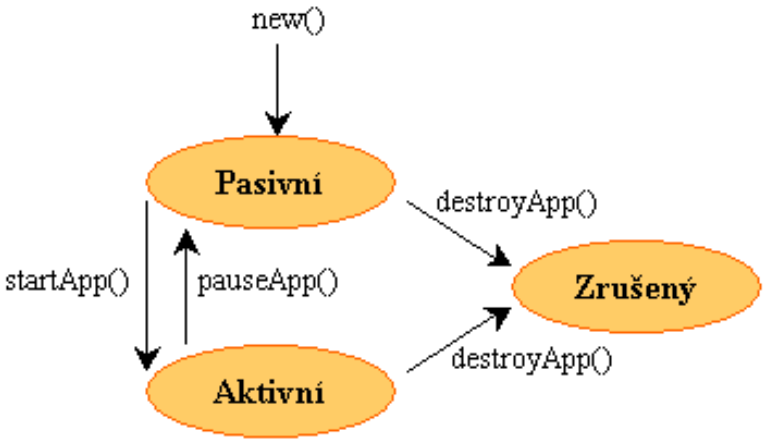
Obrázek 2: Architektura J2ME

Java ME byla navržena firmou Sun Microsystems a nahrazuje dřívejší technologii, tzv. PersonalJava. Zdrojový kód Java ME je licencován pod GNU General Public License. Java ME je rozdělena do dvou základních konfigurací, ty jsou navrženy tak, aby představovaly základní platformy pro dané zařízení a není je dále rozšiřovat. V současné době jsou definovány dvě konfigurace - CLDC a CDC.

Prvními telefony s podporou Javy byli Nokia 3410 a Siemens M50, stále zde ale byla omezení ať už v barevnosti displeje, pouze černá a bílá, nebo malém rozlišení. To byl důvod, proč některé společnosti stále podporovaly technologii WAP her. WAP začal rychle růst s příchodem telefonů s barevným displejem, schopných provádět jednoduché skripty WAP her, která byly přitažlivější v porovnání s malým monochromatickým displejem u Java her, jejich plusem ovšem byla rychlost oproti WAP hrám.

2.3.1 CLDC

CLDC neboli Connected Limited Device Configuration je určena především pro mobilní telefony nebo PDA s malou pamětí. Aplikace v CLDC konfiguraci se nazývají MIDlety. MIDlet je java aplikace postavená na základě MIDP profilu. Hlavní soubor MIDletu je soubor .jar, obsahující třídy programu, zdroje a manifest. Pro běh MIDletu na mobilním telefonu je třeba MIDlet zabalit do .jar souboru a splnit další požadavky.



Obrázek 3: Životní cyklus MIDletu [5]

2.3.2 CDC

CDC neboli Connected Device Configuration je určena k použití v zařízeních vyžadujících kompletní implementaci Java virtual machine. Zařízení s konfigurací CDC bývá výkonnější oproti zařízení s CLDC. Konfigurace CDC používá virtuální stroj nazývající se CVM (Compact Virtual Machine).

2.4 Uvedení mobilů s barevným displejem

Koncem roku 2001 nastal u mobilních telefonů grafický zlom. Ten spočíval v uvedení prvního telefonu s LCD displejem. Jednalo se o Ericsson T68, který byl vybaven pasivním LCD displejem, který byl schopen zobrazit 256 barev. Pro tvůrce mobilních her začala nová éra, již nebylo třeba omezovat se na pouhé dvoubarevné zobrazení, další výhodou byla možnost tvořit hry s 3D grafikou. Díky tomu se začaly mobilní hry čím dál tím více podobat klasickým PC hrám a získávaly si větší a větší oblibu. Postupem času se displeje stále zdokonalovaly, zvětšovalo se rozlišení, zvyšoval se počet zobrazovaných barev, měnila se technologie a v dnešní době můžeme u některých telefonů najít displeje s takovým rozlišením, které bylo před pár lety běžné na PC.

2.5 Možnosti tvorby mobilních aplikací

- Java ME: vhodné pro vertikální aplikace, které musí být přenosné, 2D grafika, mnoho widgetů
- Symbian platform: výkonné pro všeobecný vývoj, navrženo pro mobilní zařízení, pouze pro zařízení používající OS Symbian, programování v C++
- Android: Linuxové jádro, programování v jazyce Java, 3D grafika pomocí Open GL ES
- iPhone OS: používá objektové C, založené na programovacím jazyku C
- Python: ideální pro prototypy a testování konceptů
- .NET Compact Framework: vývoj pro Pocket PC, nebo Windows Mobile zařízení. Pomocí software Mono je možnost rozšíření na zařízení používající Android
- Windows Mobile: pouze Microsoft zařízení, programování buď v C/C++ nebo Basic4ppc
- Flash Lite: vhodné pro složité grafické aplikace, programovací jazyk ActionScript

3 Google Android

3.1 Systém Google Android

Android je otevřený operační systém, platforma postavena na Linuxovém jádru určená pro mobilní telefony a zařízení. Vývoj Androidu probíhá pod open source licencí. Za vznikem této platformy stojí firma Android Inc., která začala s jejím vývojem a později byla odkoupena Googlem. Později Google společně s dalšími firmami založil sdružení Open Handset Alliance, které nyní systém Android vyvijí. Pod pojmem otevřený mobilní systém si můžeme představit mobilní systém, který nám nejen dovoluje instalovat aplikace, ale narozdíl od ostatních mobilních systémů jako Symbian, Windows Mobile se liší tím, že máme možnost upravit si část nebo dokonce celý systém podle svých představ. Poslední uvolněnou verzí je verze 2.1 zvaná Eclair založená na Linux jádře verze 2.6.29. Celočíselné označení pro tuto verzi je 7, tento identifikátor označuje API úroveň a systému slouží k určení, zda-li je instalovaná aplikace kompatibilní se systémem. V této verzi mimo jiné můžeme najít Bluetooth 2.1, digitální zoom, nové uživatelské rozhraní prohlížeče s podporou HTML5, podporu pro více velikostí a rozlišení displeje a další. Kterýkoli vývojář může svou aplikaci zveřejnit na Android Marketu, kde je dostupný seznam všech aplikací pro Android. Android Market je přístupný z jakéhokoli zařízení se systémem Android a poskytuje zákazníkům možnost aplikace přímo stahovat a instalovat, také nabízí možnost aplikace hodnotit. Na následujícím obrázku můžeme vidět robotické logo Androidu.

Obrázek 4: Logo Androidu [2]

3.2 Historie Androidu

Společně se založením sdružení OHA dne 5.listopadu 2007 byl ohlášen i Android. Sdružení OHA v současné době sdružuje 65 společností a zabývá se vývojem otevřených standardů pro mobilní zařízení. Několik dní po ohlašení Androidu, konkrétně 12. listopadu 2007, OHA uvolnilo první ukázkovou verzi Android SDK 1.0, bylo v ní zahrnuto vývojové prostředí, emulátor, debugger, sada knihoven, dokumentace, ukázkové programy, tutoriály a další. Vydání první veřejně dostupné platformy bylo na počátku roku 2008. Věškeré součásti platformy jsou pod licencí „Apache free-software and open-source license“ dostupné komukoli. Prvním mobilním telefonem běžícím pod Androidem byl HTC Dream označovaný jako T-Mobile G1 podle operátor T-Mobile, který ho jako jediný ve světě prodává. Tento telefon byl představen 23.září 2008. Jedná se o smartphone, který disponuje všemi funkcemi běžnými u těchto typu telefonů. Displej je dotykový s rozlišením 480x320 pixelů, může pracovat v horizontální i vertikální poloze, je osazen

dvoujádrovým procesorem firmy Qualcomm MSM7201A o frekvenci 528 Mhz, podporujícím i 3G sítě. Odtlačením displeje získáme plnohodnotnou QWERTY klávesnici, jinak je standardně přístupných 6 tlačítek pro základní navigaci. Pro uložení dat můžeme použít kromě interní paměti také paměťovou kartu typu microSD. Datové přenosy je možno uskutečnit pomocí miniUSB, GPRS, HSCSD, EDGE a dalších, telefon rovněž podporuje Wi-Fi 802.11 b/g. T-Mobile G1 je schopen přehrávat většinu běžných hudebních formátů, z video formátů umí přehrát H.264, 3GPP, MPEG4 a 3GP. Zajímavou funkci telefonu je naklánění mapy podle aktuální polohy díky integrovaného přijímače GPS s elektromagnetickým kompasem. Následující obrázek představuje hlavní plochu Androidu po spuštění.

Obrázek 5: Hlavní plocha Androidu [3]

3.3 Vývoj aplikací v Androidu

Při psaní aplikací pro Android je možno využít řízený kód psaný v programovacím jazyce Java, což umožňuje Android SDK, a ovládat zařízení pomocí Java knihoven vyvinutých Googlem. Android ale nepoužívá pro spouštění zkompilovaných aplikací psaných v Javě standardní Java Virtual Machine, ale používá vlastní virtualní nástroj zvaný Dalvik Virtual Machine. Hlavním důvodem použití Dalviku VM je způsob jak obejít licenci Sunu a možnost používat Java knihovny. Také by měl běžet rychleji i na mobilních telefónech s omezeným hardwarem.

Software pro Android je pomocí vývojářských nástrojů možné vyvíjet pod desktopovými operačními systémy Windows, Mac OS X a Linux na hardwarové platformě x86-kompatibilní. K vývoji dále budeme potřebovat Apache Ant, Java Developement Kit a Python. Pro přidání a aktualizace SDK komponent se používá Android SDK a AVD Manager, pomocí něhož lze přidat různé platformy Androidu, nové vývojové nástroje, novou dokumentaci a další. Nové SDK komponenty jsou automaticky instalovány do existujícího SDK adresáře, takže není nutné aktualizovat vývojové prostředí, kvůli novému umístění SDK. Při vývoji aplikací pro Android máme na výběr z více variant vývojových

prostředí. Oficiálně podporovaným vývojovým prostředím je Eclipse, do kterého je třeba nainstalovat rozšíření Android Developement Tools Plugin. Díky tomu můžeme vytvářet a ladit naše Android aplikace snadněji a rychleji. Spojením Eclipsu s ADT pluginem dostáváme silný nástroj pro vývoj:

- Přístup k dalším Android vývojovým nástrojům uvnitř Eclipse. Např. přístup k mnoha možnostem DDMS nástrojů: nastavení breakpointů, vytvoření screenshotu a zobrazení vláken a informací o procesu přímo z Eclipsu.
- Poskytnutí průvodce nového projektu, který napomáhá vytvoření a nastavení základních souborů potřebných pro novou Android aplikaci
- Automatizuje a zjednodušuje proces sestavení Android aplikace
- Poskytuje Android editor kódu, který pomáhá při psaní validních XML pro Android manifest a zdrojových souborů

Kromě Eclipsu lze samozřejmě použít i jiné vývojové prostředí, např. NetBeans, které ovšem není oficiálně podporováno Googlem. Vývojářům rovněž zůstává možnost použít pro vytváření, kompilování a ladění svých aplikací nástroje příkazového řádku. Máme možnost si přizpůsobit naše vývojové prostředí, díky tomu, že každá verze Android platformy může být nainstalovaná jako jednotlivá složka našeho SDK. Pomocí AVD manageru si můžeme nastavit požadované nastavení Androidovského virtuálního zařízení, aby co nejvíc odpovídalo cílovému zařízení, tímto si následně můžeme ověřit kompatibilitu vytvořených aplikací aniž bychom vlastnili dané zařízení. Každé takovéto zařízení je složeno z:

- Hardwarového profilu. Umožňuje nastavení hardwarových funkcí, např. jestli je povolená kamera, velikost paměti, použitou klávesnici atd.
- Mapování systémového obrazu, definování jaké platforma poběží na zařízení
- Další nastavení, třeba vzhled emulátoru, velikost displeje atd.
- Vyhrazeného prostor pro uložená data na cílovém zařízení

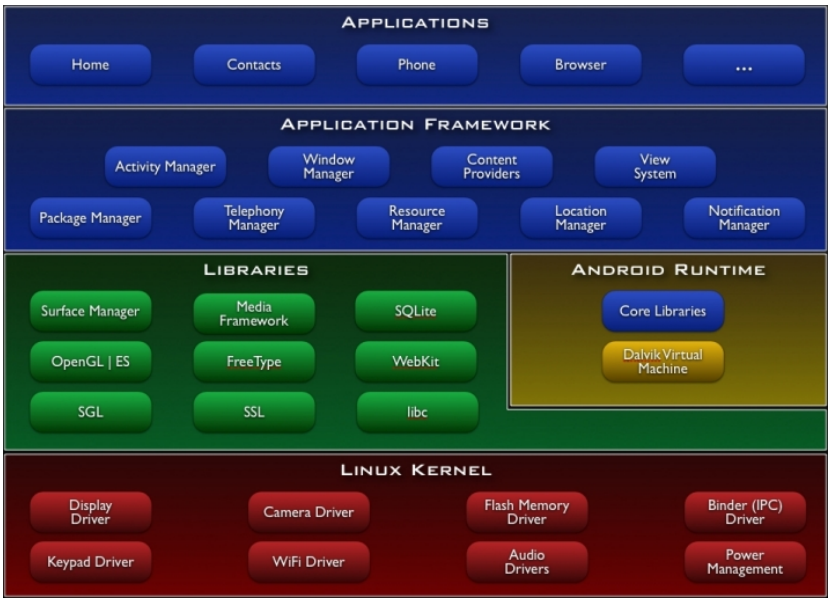
Je možné vytvořit několik takových zařízení s různým nastavením, podle požadovaného modelu a typu platformy.

Jak již bylo zmíněno výše Android používá pro spuštení zkompilovaného kódu vlastní Dalvik Virtual Machine. Dalvik byl napsán Danem Bornsteinem a pojmenován podle místa dokončení Dalvík na Islandu. Oproti běžným virtuálním strojům, které jsou realizovány jako zásobníkové stroje, Dalvik VM je architektura založena na registrech. Obecně zásobníkové stroje musí používat instrukce pro načtení dat na zásobníku a manipulaci s těmito daty a proto požaduje více instrukcí než stroje používající registry pro implementaci stejného kódu, ale instrukce ve strojích používajících registry musí zakódovat vstupní a cílové registry proto má tendenci být větší. Pro konverzi Java .class do .dex formátu, vhodného pro systémy, které jsou omezeny rychlosti procesoru a pamětí, se

používá nástroj zvaný dx. Více tříd je zahrnuto do jediného .dex souboru. Stejně tak Java bytecode je převedena na alternativní instrukční sadu používanou Dalvik VM. Spustitelný soubor Dalvik může být znova upraven, když byl naistalován na mobilní zařízení. Dalvik VM má některé specifické vlastnosti pro optimalizaci při požadavcích na malou paměť:

- Virtuální stroj byl zredukován
- Dalvik nemá JIT překladač
- Pro zjednodušení interpretru byl upraven zásobník konstant pro použití pouze 32-bitových indexů
- Používá vlastní bytecode místo Java bytecode

Pokud naši aplikaci testujeme na emulátoru můžeme pomocí služeb Androidu samozřejmě vyvolat další aplikace, přistoupit k síti, přehrát hudbu nebo video, příjmout a odeslat data a další. Emulátor obsahuje také řadu různých ladících nástrojů, jako konzoli ze které můžeme zaznamenávat výstupy jádra, simulovat přerušení aplikace apod. Spoustu užitečných informací ohledně vývoje aplikací pro Android je možno nalézt na adrese http://developer.android.com. Na obrázku je vyobrazena architektura Androidu.



Obrázek 6: Přehled architektury Androidu [4]

4 Grafické možnosti tvorby na platformě Google Android

Při tvorbě grafiky nám Android nabízí možnost využít buď běžné grafické knihovny pro 2D grafiku nebo OpenGL ES 1.0 pro výkonné 3D grafické aplikace.[2]

4.1 2D grafika

Nejpoužívanější API pro 2D grafiku se nachází v balíku drawable. API pro OpenGL jsou dostupná přes balík Khronos OpenGL ES a další Android OpenGL nástroje. Pokud vyvíjíme aplikace s 2D grafikou, máme na výběr ze dvou možností:

- Vykreslit grafiku nebo animaci do View objektu našeho Layoutu,
- vykreslit grafiku přímo do Canvasu.

4.1.1 Vykreslení pomocí Layout

Při výběru první možnosti je vykreslování grafiky řízeno klasickou View hiearchií vykreslovacího procesu - jednoduše se definuje grafika, která se bude vyobrazova pomocí View. Tato volba je nejlepší pro vykreslení jednoduché grafiky, kterou není potřeba dynamicky měnit, například zobrazení statického grafu nebo předdefinované animace.

4.1.2 Vykreslení s použitím Canvasu

Pokud zvolíme druhou možnost budeme volat příslušné *draw()* metody nebo jednu z Canvas *draw()* metod. Použití této možnosti je dobré především, pokud potřebujeme pravidelně překreslovat plochu. Jakékoli videohry by měly být vykreslovány na vlastní Canvas, pro to máme opět 2 způsoby:

- Volat metodu *invalidate()* a ovládat událost *onDraw()* pro vytvořenou View komponentu v našem layoutu a stejném vlákně jako aktivita uživatelského rozhraní,
- nebo v odděleném vlákně, kde ovládáme `SurfaceView` a provádět překreslování Canvasu tak rychle, jak jen to vlákno umožňuje. Je zde možnost i přizpůsobit FPS, díky čemuž můžeme ušetřit baterii.

Práci Canvasu si můžeme představit jako rozhraní opravdové plochy na kterou bude naše grafika vykreslována, obsluhuje všechny naše „draw" události. Pomocí Canvasu můžeme na plochu umístit nějaký základní obrázek, resp. Bitmapu. V případě vyvolání události *onDraw()*, stačí zavolat metodu pro vykreslení Canvasu. Pokud se zabýváme SurfaceView objekty, je možné vyžádat Canvas pomocí metody *SurfaceHolder.lockCanvas()*. Pokud chceme vytvořit nový Canvas, musíme definovat `Bitmap`u, která bude vykreslena. Bitmapa je pro Canvas vždy vyžadována.

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

Výpis 1: Ukázka nastavení Bitmapy

Nyní může Canvas vykreslovat na tuto Bitmapu. Bitmapu je možné použít i s jiným Canvasem, přeneseme ji pomocí jedné z metod *Canvas.drawBitmap(Bitmap,...)*. Pro konečné vykreslení naší konečné grafiky pomocí Canvasu se doporučuje použít metody *View.onDraw()* nebo *SurfaceHolder.lockCanvas()*. Ve třídě Canvas můžeme nalézt několik vykreslovacích metod, které můžeme použít, například *drawBitmap(...)*, *drawRect(...)*, *drawText(...)* a další. Také v jiných třídách můžeme nalézt *draw()* metody. Například třída Drawable má vlastní metodu *draw()*.

4.1.2.1 View Použití View je spíše pro aplikace, které nepožadují velké množství zpracování. Výhodou této metody jsou předdefinované Canvasy, které nám Android framework nabízí. Při použití View musí naše třída dědit z této třídy a definovat metodu *onDraw()*. Zde se budou provádět veškerá volaní pro kreslení skrz Canvas. Android vyvolá tuto metodu, pokud je požadováno vykreslení View a bude volána pouze podle potřeby. Pokud je naše aplikace připravena vykreslit vyvoláme metodu *invalidate()*, která indikuje, že chceme dané View vykreslit a prohlásí ho za neplatné.

4.1.2.2 SurfaceView Je specialní potomek třídy View poskytující vyhrazený prostor pro vykreslování s View hierarchií. Při této variantě je vykreslování obsluhováno ve druhém vlákně, takže aplikace nemusí čekat, dokud hierarchie View není připravena na vykreslení. Pro začátek je třeba vytvořit třídu dědící ze `SurfaceView` a také by měla implementovat `SurfaceHolder.Callback` toto rozhraní nás bude upozorňovat o stavech Surface, například vytvoření, změna nebo zrušení. Tyto události jsou důležité, abychom vědeli, kdy máme začít vykreslovat, nebo provést nějaké změny a kdy přestat vykreslovat. Objekty Surface bychom měli ovládat pomocí `SurfaceHolder`, ten získáme pomocí metody *getHolder()*. Aby bylo možné vykreslit Canvas z druhého vlákna je třeba si ho zamknout pomocí metody *lockCanvas()*. Po vykreslení Canvasu je nutné zavolat metodu *unlockCanvasAndPost()* pro uvolnění a zveřejnění Canvasu. Surface nyní vykreslí tento Canvas. Sekvenci zamykání a odemykání budeme vykonávat pokaždé, když budeme požadovat překreslení.

4.2 3D grafika

Jak už bylo zmíněno výše Android poskytuje podporu pro výkonné 3D aplikace přes OpenGL API, přesněji OpenGL ES API. OpenGL ES je „osekaná“ verze OpenGL ES, určená pro mobilní zařízení. Android nyní podporuje verzi OpenGL ES 1.0, která odpovídá verzi OpenGL 1.3. API poskytované Androidem je podobné J2ME JSR-239 OpenGL ES API.

5 Návrh a implementace shoot'em'up hry

Zadaní mé bakalářské práce bylo navrhnout herní jádro a jednodušší hru pro mobilní telefon postavenou na tomto jádře. Pro tento návrh jsem si zvolil mobilní operační systém Android. Hra by měla být typu shoot'em'up. „Shoot'em'up je žánr počítačové hry, kde má hráč omezenou kontrolu nad svojí figurkou (obvykle letadlo nebo vesmírná loď) a důraz je většinou kladen na zničení nepřítele.“ [5]. Spojením těchto dvou faktorů vznikl pracovní název hry, kterou jsem pojmenoval *AndroSh00ter*. V následujících částech budu popisovat postup vývoje a hru samotnou.

5.1 Popis hry

AndroSh00ter je v podstatě letecká střílečka, pro jednoho hráče, který ovladá svoji loď a snaží se sestřelit co nejvíce nepřátelských lodí, což je cílem této hry. Při tvorbě hry jsem se nechal inspirovat hrami jako Space Blaster, Raptor apod. Hru jsem koncipoval pouze jako „portrét“, tzn. na výšku. Hra má zatím celkem 10 levelů, které bude možno editovat na PC. Ve hře se objevuje 5 druhů nepřátelských lodí, kde každá má jiné vlastnosti a jiné ohodnocení. Každý level obsahuje jiné nepřátelé, případně jiné vlastnosti nepřátel. Po dosažení určitého skóre bude hráči nabídnut jeden ze 4 speciálních pomocných prvků.

Obrázek 7: Ukázka lodi hráče

Po spuštění hry si můžeme z menu vybrat buď novou hru pomocí `Spustit hru`, nebo vybrat uloženou hru pomocí `Načíst hru`. V menu ještě můžeme nalézt položky `Highscore` a `Nápověda`. V menu nenajdeme položku Konec hry, OS Android má totiž vlastní algoritmus, kterým si sám řídí ukončování aktivit, proto pro opuštění hry stačí stisknout tlačítko „zpět“, případně „Home“. Hra se bude ukládat automaticky po ukončení, pokud k tomu ovšem nedošlo vyčerpáním životů. Těch má hráč k dispozici celkem 3 a v průběhu hry může jeden získat. Hráčova loď se může pohybovat 4 směry: doprava-doleva a nahoru-dolů, přičemž hranice tvoří šířka a výška displeje. Pro ovládaní lodě jsou určeny především klávesy D-Padu, hra ale podporuje i jednoduché dotykové ovládání.

5.2 Průběh vývoje hry

Po vybrání typu hry, bylo třeba vybrat způsob jakým hra bude prezentována. Jelikož má mít hra pouze pseudo 3D grafiku, rozhodl jsem se využít běžné 2D grafické API místo výkonného OpenGL pro 3D aplikace. Grafické zobrazení se bude vykreslovat přímo do Canvasu, k tomuto se nabízela jedna z možností, využít View nebo SurfaceView. Rozhodl jsem se pro první variantu, protože se jedná se o jednoduchou hru a není třeba

používat další vlákno pro výpočty, v případě složitějších výpočtu by byla druhá varianta výkonnější.

5.2.1 Menu

Třída Menu je spouštěcí třídou, ze které se budou spouštět ostatní položky. Rozložení položek menu je načítáno ze souboru main.xml, který se nachází ve složce s *layouty*. Položky menu mají 3 stavy:

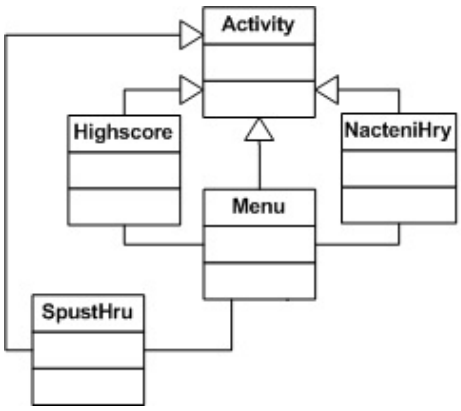
- 

neaktivní - po spuštění hry
- 

vybraná - po najetí na položku
- 

stisknutá - po stisknutí položky

Vybraná a stisknutá může být vždy jen jedna položka, zbytek je neaktivní. Pokud vybereme jednu z položek menu, spustí se nová aktivita. Položky menu jsou realizovány jako tlačítka v TableLayout, která při obsluze události vyvolávají spuštění vybrané aktivity. Každá položka resp. tlačítko má svůj vlastní XML layout, kde je definovaný vzhled těchto položek. Všechny aktivity, které chceme spouštět musí být uvedeny v souboru AndroidManifest.xml. [6]



Obrázek 8: Základní diagram tříd

Jak můžeme na diagramu uvedeném výše vidět, třída `Menu` dědí ze třídy `Activity` a obsahuje následující třídy:

- `SpustHru` - spuštění samotné hry
- `NacteniHry` - načtení uložené hry
- `Highscore` - zobrazení 5 nejlepších výsledků

```
Button StartGameButton = (Button)findViewById(R.id.SpustHru);
        StartGameButton.setOnClickListener(new OnClickListener() {

         public void onClick(View v) {
             Intent  StartGameIntent = new Intent(Menu.this,SpustHru.class);
              startActivity  (StartGameIntent);
         }
        });
```

Výpis 2: Ukázka spuštění nové aktivity z menu

5.2.2 Highscore

Samotná třída `Highscore` je potomkem Activity a slouží pouze k nastavení obsahu View pomocí třídy `VypisHighscore`. Tato třída je dědená z View a slouží k výpisu 5 nejlepších dosažených skóre a případně jejich smazání. Highscore je uloženo v souboru *highscore.dat*. Čtení tohoto souboru se provádí ve třídě `Zapisovac`, která slouží i k zapisování highscore po ukončení hry. Načtení je realizováno pomocí metody *openFileInput(String name)*, jíž předáváme atribut s názvem souboru a můžeme ji najít ve třídě `Context`. Data jsou načtena do bufferu a následně předána k dalšímu zpracování jako String.

```
public String  nactiHighscore(Context context){
        FileInputStream fIn  = null;
        InputStreamReader isr = null;

        char[]  inputBuffer  = new char[255];
        String  data = null;

        try{
         fIn  = context.openFileInput("highscore.dat");
             isr  = new InputStreamReader(fIn);
             isr .read(inputBuffer);
             data = new String(inputBuffer);
             vypIn(data);
             }
            ...
             return data;
       }
```

Výpis 3: Ukázka metody pro načtení souboru s highscore

Po načtení hodnot ze souboru se ve třídě `VypisHighscore` uloží hodnoty do pole typu long a jména do pole Stringů. Tyto údaje se budou následně v cyklu vykreslovat do canvasu pomocí metody *drawText(String text, float x, float y, Paint paint)*. Pomocí instance třídy `Paint` lze textu nastavit různé vlastnosti, například font, velikost, barvu a jiné. Třída `Zapisovac` také obsahuje metodu *deleteFile("highscore.dat")* třídy `Context` pro mazání souboru s highscore. Další využití třídy `Zapisovac` bude uvedeno v následujících kapitolách.

5.3 Implementace hry

V této sekci budou popsány třídy, definující přímo chování a další aspekty hry.

5.3.1 Třída SpustHru

Třída `SpustHru` je opět potomkem třídy `Activity` a je asi nejdůležitější třídou celého projektu. Pomocí této třídy se spouští samotná hra. Abychom mohli využít co největší prostor displeje, zakázal jsem pomocí metody *requestWindowFeature(int featureId)* zobrazování titulku. Kvůli možnosti obsloužit události vyvolané například příchozím hovorem a podobně, si třída drží reference na layout, které se používají k pozastavení hry pomocí metod *onStop(), onPause(), onRestart()* třídy `Activity`. Po zavolání *onPause()* se akorát zavolá metoda *onStop()*, která zastaví časovač, jenž je používán jako herní smyčka. Po spuštění třídy SpustHru se View nastaví na jednoduchý layout pro zadání jména hráče.

Po zadání jména se nastaví View definované pomocí XML layout. Pro inicializaci tohoto XML souboru jsem použil statickou metodu *inflate(int resource, ViewGroup root)* třídy `LayoutInflater`, která vrátí View objekt. Tato metoda se skládá z parsování XML layout souboru, vytvoření odpovídajícího View objektu a jeho přidání do view hierarchie.

```
<?xml version="1.0" encoding="utf−8"?>
<com.Tutorial.MojeHra xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/AndrooGame"
    android:orientation=" vertical "
    android:layout_width=" fill_parent "
    android:layout_height=" fill_parent "
    android:background="#FF000000">
</com.Tutorial.MojeHra>
```

Výpis 4: Ukázka layout souboru spusthru.xml

Na prvním řádku se nachazí klasická hlavička XML souboru. Pro nás je důležitý druhý řádek, kde najdeme kořenový element, který určuje layout dokumentu. Do kořenového elementu lze přidávat další layout objekty nebo další potomky, které budou definovat layout. V našem případě je layout nastaven na třídu `MojeHra`, která je potomkem abstraktní třídy `Andro` dědící ze třídy `LinearLayout`. Popis obou těchto tříd bude následovat.

Parametrem *id* se nastaví jednoznačná identifikace prvku, který pak můžeme najít v automaticky generované třídě `R` a jednoduše k němu přistoupit. Po kompilaci aplikace je toto *id* referencováno jako integer, i když v layout XML bývá povětšinou jako string. Dalším parametrem je *orientation* který určuje polohu displeje, v tomto případě vertikální, tudíž „na výšku“. Dalšími parametry jsou *layout_width a layout_height* pomocí nichž se nastaví šířka, resp. výška View, v tomto případě se bude vyplňovat celá šířka i výška rodiče. Posledním parametrem je nastavení barvy plochy. Následně toto View nastavíme jako obsah pro vyobrazení a pomocí metod *setFocusable(true) a setFocusableInTouchMode(true)* povolíme interakci s uživatelem pomocí klávesnice. Pokud bychom toto vynechali, nebylo by možné obsloužit události vyvolané stisknutím kláves.

5.3.2 Třída Andro

`Andro` je abstraktní třída, která je potomkem třídy `LinearLayout`. Pomocí této třídy definujeme layout pro View. Všechny uživatelsky definované layouty musí definovat konstruktor Android `Context` a sadu atributů. Dále jsou zde definované abstraktní metody, které budeme definovat v odvozené třídě `MojeHra`, například *inicializace*. V této třídě je potřeba přepsat metodu základní třídy *onLayout(boolean changed, int l, int t, int r, int b)*, která se volá v případě, že je potřeba nastavit rozměry a umístění všech potomků View. V této metodě se bude volat *inicializace()* a *spustCasovac()*. Abstraktní metoda *inicializace()* slouží k počátečnímu nastavení jednotlivých levelů, její podrobná implementace bude uvedena ve třídě `MojeHra`.

Druhá zmíněná metoda *spustCasovac()* spustí časovač, který slouží k udržení herní smyčky. V případě přerušení hry jinou aktivitou se časovač zastaví a po návratu zpět se opět spustí. Tento časovač periodicky ve zvoleném čase vyvolává událost, která následně spouští aktualizaci jednotlivých objektů.

```
protected void spustCasovac() {
    _casovac = new Timer();
    _casovac.schedule(new Aktualizuj(), 0, _perioda);
}
```

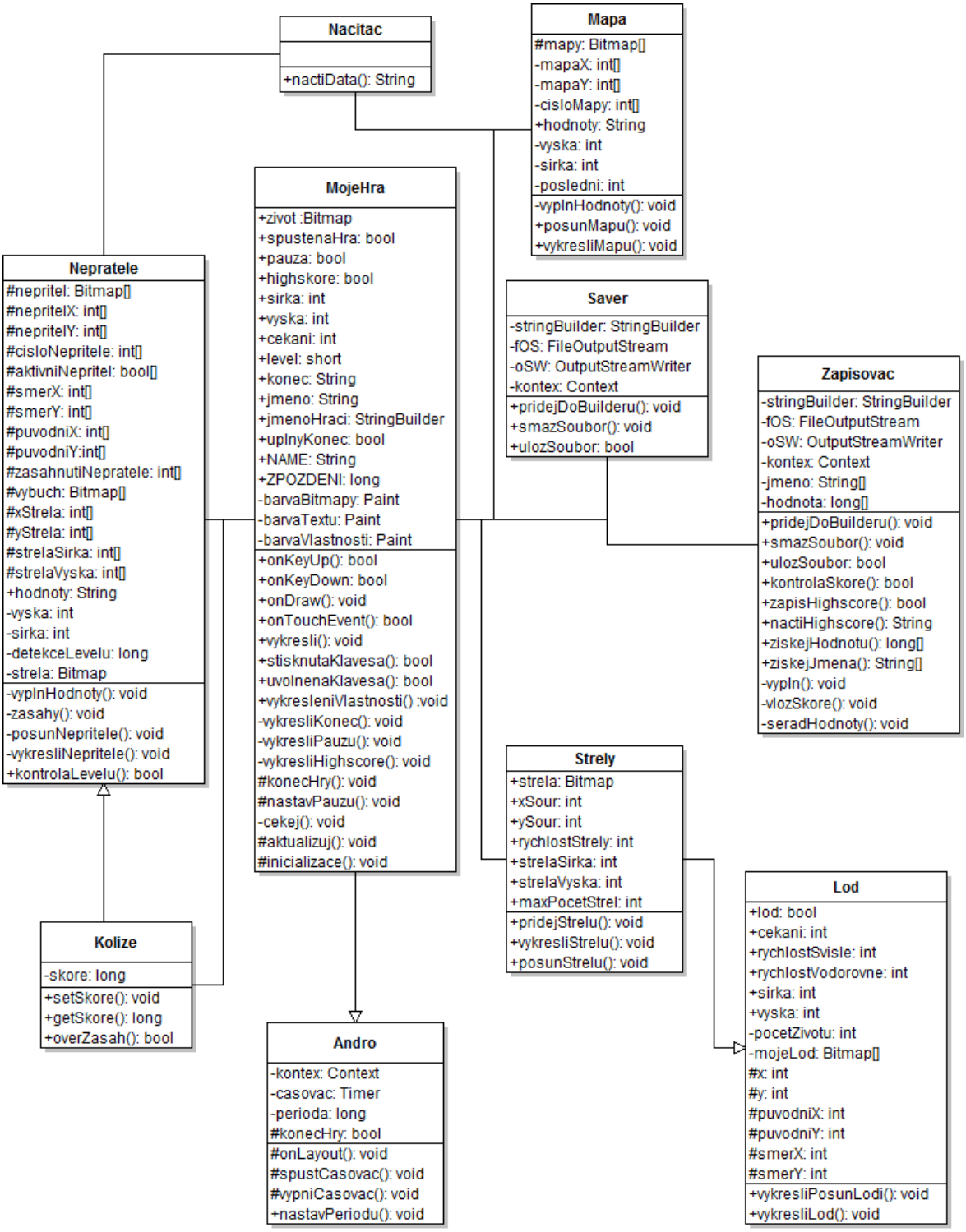
Výpis 5: Definice časovače

To se provádí ve třídě `Aktualizuj` odvozené ze třídy `TimerTask` v metodě *run()*, spouštějící samotnou aktualizaci. Tímto způsobem byla vytvořena jednoduchá obnovovací smyčka. Poté je třeba zavolat metodu *invalidate()* třídy `View`, která zruší současné View, aby mohlo být vykresleno a nahrazeno novým. Jelikož je toto volání metody prováděno mimo vlákno uživatelského rozhraní je nutno použít metodu *postInvalidate()*. Tímto docílíme vyvolání metody *onDraw(Canvas canvas)*, vykreslující veškeré námi nastavené objekty. Třída `Canvas` udržuje veškerá volaní pro vykreslení. Pro vykreslení čehokoli je potřeba těchto základních věcí: Bitmapu udržující pixely, vykreslovaný typ, např.text, cesta nebo Bitmapa, Canvas obsluhující volání vykreslování a Paint popisující barvu a styl pro kreslení.

Poslední metodou obsaženou v této třídě je *nactiObrázek(int id)*, vracející obrázek jako Bitmapu, kde „id“ je identifikační číslo obrázku, které odpovídá obrázku uvedeném v souboru `R.java` a uloženém ve složce `res\drawable` . Pro převedení obrázku na Bitmapu se používá statická metoda *decodeResource (Resources res, int id)* nacházející se ve třídě `BitmapFactory`.

5.3.3 Třída MojeHra

Ve třídě `MojeHra` se nacházejí veškeré důležité prvky naší hry a vytváří logiku hry. Tato třída je centrem pro naše jádro. Je odvozená od abstraktní třídy `Andro` a proto jsou zde definovány všechny abstraktní metody rodiče. Třída `MojeHra` si uchovává instance na objekty všech ostatních tříd tvořících jádro a probíhá zde počáteční inicializace AndroSh00teru.



Obrázek 9: Diagram tříd tvořící herní jádro

Mezi nejdůležitější třídy tvořící jádro patří `Nepratele`, obsahující metody a hodnoty týkající se nepřátel, potomkem této třídy je třída `Kolize`, které zahrnuje metodu pro detekci kolizí nepřátel s dalšími objekty, dále zde patří třída `Lod`, které uchovává všechny potřebné vlastnosti hráčovy lodě a dědí z ní třída `Strela` zahrnující vlastnosti a nastavení týkající se střel. Dědičnost u těchto dvou tříd je použita z důvodu snadnějšího získaní pozice, ať už lodě nebo nepřátel. Poslední důležitou třídou je třída `Mapa`, obsahující vlastnosti a další aspekty tvořící podkladovou mapu. Ta je v tomto případě složená z různých ostrůvků, které ovšem neovlivňují samotnou hru. Třídy `Zapisovac` a `NacitacSouboru` jsou pomocné třídy sloužíci k načtení a zápisu highscore, resp. načítání nastavení levelů a map ze souborů. Všechny zmíněné třídy budou popsány v následujících odstavcích.

V konstruktoru třídy `MojeHra` nastavíme periodu časovače na 50 milisekund. Zavedeme proměnné pro indikaci stavu hry:

- spustenaHra - indikace, jestli už hra byla spustěna, počáteční stav je false
- pauza - indikace pauzy, posuny všech objektů jsou zastaveny, počáteční hodnota false
- konecHry - indikuje konec hry a čeká na potvrzení
- highscore - neslouží k indikaci skóre, ale oznamuje, jestli jsme se dostali se svým skóre mezi 5 nejlepších

Inicializace hry spočívá hlavně v nastavení všech počátečních hodnot a načtení obrázku do Bitmap, které budou představovat objekty ve hře. Nyní si metodu *inicializace()* podrobněji popíšeme. Metoda *inicializace()* se bude spouštět při každém spuštění hry, tj. i při návratu po přerušení jinou aktivitou. Mimo to se bude *inicializace()* spouštět i při každém novém levelu. Jako první se nastaví uživatelská výška a šírka displeje, tyhle hodnoty budou následně sloužit jako hranice při vykreslování prvků, aby nedocházelo k vykreslování Bitmap mimo viditelnou oblast.

Barva představuje instanci třídy `Paint`, sloužící k uchovávání informací o stylu a barvách, které boudou použity pro kreslení nejen obrazců. Je možné nastavit, třeba barvu pozadí, font písma, jeho velikost písma a další. Barvu pro Bitmapy jsem zvolil v odstínu modré, připomínající barvu vody. Poté je nastavena barva textu a jeho vlastnosti, bude se vykreslovat tučně a počáteční velikost je nastavena na 12px. Poslední nastavovanou barvou je barva pro „stavový“ řádek, ve kterém se bude zobrazovat počet životů, aktuální skóre a číslo aktuálního levelu. Tento řádek bude umístěn u dolního okraje displeje, uživatelská část displeje proto bude o tento řádek zkrácena.

Následuje nastavení všech Bitmap používaných v této hře. V případě, že je Bitmap pro určité zobrazení více vložíme si jejich id do pole, tyto parametry budou sloužit ke získání obrázků pomocí metody *nactiObrazek(id)*. Nakonec nastavíme všechny používané instance. Instance třídy `kolize` je nastavována v try-catch bloku z důvodu načítání dat ze souboru, což může vyvolat výjimku, pokud je soubor nenalezen, nebo poškozen. Poslední věcí v inicializaci je spuštění jakéhosi „zpožďovače“, který zpozdí o pár sekund

samotné spuštění hry, aby se mohl hráč připravit. Po provedení inicializace se spustí časovač, který udržuje herní smyčku, jakým způsobem je popsáno výše.

Jak jsem již dříve zmínil metoda *run()* vyvolávaná časovačem slouží k aktualizování hodnot a následné indikace vykreslování. Časovač pro tuto metodu spouští ve vlákně. Po prvním spuštění aktualizace se čeká až se spustí hra, tato hodnota je nastavena na 50 a dekrementuje se při každém průchodu aktualizací dokud není 0. Po dosažení 0 se hra spustí. Kompletní herní smyčka je realizována v pořadí:

1. Posun mapy, posouvá všechny ostrovy, tvořící podkladovou mapu,
2. posun nepřátel, posouvá všechny nepřátelé a kontroluje jejich případné kolize, pokud je zjištěna kolize, nepřítel se označí za „neviditelného“ a na jeho místě se vykreslí výbuch,
3. posun lodě, posouvání lodě podle ovládání kláves nebo pomocí tažení a detekce případné kolize s nepřítelem,
4. posun střel, posouvání všech vyslaných střel, pokud střela proletí bez zásahu nebo zasahné nepřítele bude odebrána,
5. kontrola levelu, kontroluje, zda-li už jsme nedosáhli dalšího levelu a není třeba jej načíst,
6. kontrola životů, zjišťuje, jestli už jsme nevyčerpali veškeré životy, pokud ano, vyvolá konec hry,
7. vykreslení všech prvků podle jejich pozice.

Ve stavovém řadku nalevo jsou informace o levelu a skóre, vpravo pak zobrazení životů. Loď je možno kromě D-padu, ovládat i tlačítky Q pro směr vlevo, W pro směr vpravo, O směr nahoru a L směr dolů. Střely se vysílají buď to mezerníkem nebo prostředním tlačítkem D-padu. Klávesy P slouží k dočasnému zastavení hry, je vypsán informativní text a možnost návratu do menu, tím se ovšem hra ukončí.

Obrázek 10: Zobrazení režimu pauza

Při pozastavené hře nedochází k posouvání objektů, tyto jsou navíc překresleny namodralou „mlhou“, jak můžeme vidět na obrázku. Pauzu lze zrušit opětovným stisknutím P. Loď je zobrazována ve 3 režimech, nakloněná doprava, nakloněná doleva a horní pohled. Při výstřelu střela vyletí buď ze středu lodě, nebo v případě bonusového střílení se budou vystřelovat 2 střely z okrajů lodi. V prvních kolech je povoleno pouze 12 aktivních střel, s přibývajícími koly se počet střel zvedá. Abychom měli střelbu ztíženou není možné střílet ze všech pozic. Délka levelu je dána počtem nepřátel. Za každého zasaženého nepřítele se přičítá skóre v rozmezí 100 - 500 bodů. Životy ubývají po srážce s nepřítelem. Po jejich vyčerpání se vyvolá konec hry, ten je označen informativní zprávou o dosaženém skóre a informací, jestli jsme se dostali mezi 5 nejlepších. Do menu se pak můžeme vrátit klávesou zpět. Reakce na události vyvolané stiskem klávesy nebo pomocí dotykového módu jsou řešeny přepsáním následujících metod:

- onKeyDown: vyvolána při stisku klávesy, přijímá 2 argumenty a to kód klávesy a KeyEvent. Kód klávesy představuje celé číslo, pro zjištění o jakou klávesu se jedná tento kód jednoduše porovnáme se statickou konstantou klávesy uloženou v KeyEvent. Tato metoda vrací true, pokud chceme klávesu obsloužit pomocí nějakým naším způsobem, nebo vrátí false, pokud chceme nechat obsloužit jiným způsobem, převážně systémem. Tohoto využívám při ukončení aplikace, kdy klávesa zpět je obloužena systémem, ostatní klávesy jsou obslouženy pomocí metody *stisknutaKlavesa(int kodKlavesy)*.

- onKeyUp: událost vyvolaná při uvolnění klávesy, platí pro ni téměř to samé, co u onKeyDown. Po vyvolání této metody volám metodu *uvolnenaKlavesa(int kodKlavesy)* ve které se především nastaví pohyb lodi na 0, případně zruší střílení.

- onTouchEvent: vyvolá se při dotyku plochy a přijímá `MotionEvent`. Má taky 2 stavy, podobně jako kláves. Pomocí MotionEvent lze získat souřadnice x a y. Při použití dotykového módu je loď možno přemisťovat tažením, při poklepání vystřelí.

Použité podkladové mapky jsou tvořeny několika obrázky ostrovů jejichž souřadnice jsou uloženy v souboru mapyX.dat, kde X označuje číslo levelu. Pozadí pro mapky je nastavenou na modrou, která představuje vodu, tímto je dotvářena iluze, že letíme nad vodou. V konstruktoru `Mapy` je přijímáno pole Bitmap, představující ostrovy a výška uživatelského displeje, pro určení nové pozice mapy.

V konstruktoru se nastaví instance na třídu `NacitacSouboru`, pomocí jejíž metody *nactiData* získáme vlastnosti jednotlivých ostrovů ze souboru ve formě řetezce. Tyto hodnoty jsou v souborech uloženy ve formátu „x,y,číslo ostrova“ a každá trojice je oddělena středníkem. Tento řetezec si pomocí metody *split(";")* třídy `String` rozdělíme do pole, kde v každém poli bude jedna trojice. Velikost toho pole nám udává počet ostrovů a podle toho taky bude nastavena velikost polí se souřadnicemi ostrovů. Pro převod této trojice do příslušných hodnot slouží metoda *vyplnHodnoty*, ta následně prochází tímto polem a každou trojici rozdělí a vloží do příslušného pole. Aby soubor nebyl zbytečně moc velký a načítaní netrvalo dlouho, jsou obě souřadnice uloženy v kladných hodnotách, při vkládání do pole se proto y-ová souřadnice zneguje. Další úspora velikosti souboru je v počtu definovaných ostrovů, stačí nadefinovat několik ostrovů a ty se poté budou cyklicky opakovat, proto si musíme udržovat číslo posledního ostrova. Za tento ostrov se budou v určitém odstupu vkládat další ostrovy, které už prošly celou obrazovkou, tím bude zabezpečeno, že tyto ostrovy nikdy nepůjdou přes sebe, pokud už tak nebyly definovány.

Mapa se posouvá rychlostí jeden pixel každou aktualizaci, je to nejpomalejší prvek, čehož využijeme při paralaxovém posouvání. Paralaxní posouvání nebo taky paralaxní scroling „je jedna z technik v počítačové grafice, která simuluje 3D hloubku pohyblivého obrazu ve 2D. Používají se zejména v oblasti počítačových her, kde se tento žánr nazývá scrolling shooter.“ [5] AndroSh00ter patří mezi vertikálně skrolované hry. Ve hře AndroSh00ter je obraz skládán z 5 vrstev:

- statické modré pozadí
- vrstva s ostrovy
- vrstva s nepřáteli
- vrstva s lodí hráče
- vrstva se střelami

Různá rychlost každé vrstvy nám vytváří výše zmíněný scrolling shooter. Ve třídě definující loď hráče najdeme soukromou proměnnou uchovávající počet životů. Přístup k této proměnné je řešen pomocí vlastností. Je definována i vlastnost pro nastavení, které se využije v případě načtení uložené hry. Souřadnice lodi budou ukládány v chráněném režimu, aby k nim bylo možno jednoduše přistoupit ve třídě `Strela`, jež je potomkem

třídy `Lod`. Třída si také uchovává původní souřadnice pro případ, že by další posun lodi byl už přes okraj. V tomto případě se souřadnice lodi nechají na původní hodnotě.

```
/**
 * konstruktor Lode
 * @param lode bitmapy s obrazky lodi
 * @param sirka sirka displeje
 * @param vyska vyska displeje
 * @param pocetZivotu pocet zivotu
 **/
public Lod(Bitmap[] lode, int sirka, int vyska, int pocetZivotu){
    _mojeLod = lode;
    _lod = true;
    _cekani = 15;
    _vyskaLode = _mojeLod[0].getHeight();
    _sirkaLode = _mojeLod[0].getWidth();
    _sirka = sirka;
    _vyska = vyska;
    _x = (sirka - _mojeLod[0].getWidth())/2;
    _y = (vyska - _mojeLod[0].getHeight());
    _pocetZivotu = pocetZivotu;
}
```

Výpis 6: Konstruktor třídy Lod

Konstruktor `Lodě` přijímá mezi parametry pole Bitmap s 3 typy lodí, šířku a výšku displeje a počet životů. Počáteční souřadnice Lodi jsou nastaveny na střed šířky a dolní okraj uživatelského displeje. Také zde najdeme indikátor, jestli je loď aktivní. Ten se využívá po kolizi s nepřítelem, kdy jsou souřadnice lodi nastaveny na počáteční hodnoty a loď určitý čas problikává. Pomocí příslušných proměnných je možné zvlášť určit rychlost lodi v horizontálním i vertikálním směru.

Odvozená třída `Strela` přijímá v konstruktoru navíc oproti rodiče ještě vlastní Bitmapy a rychlost střely, díky čemuž můžeme definovat různě rychlé střely nebo je za jejich letu zrychlovat. Po stisku mezerníku nebo středního tlačítka D-Padu se zavolá metoda *pridejStrelu()*, která realizuje střílení. Souřadnice střel jsou udržovány z důvodu jednoduchosti v seznamu typu ArrayList.

Střela se přidá následujícím způsobem, nejprve se zjistí směr lodi, je to z důvodu různých lodí a my chceme, aby naše střela vyletěla ze středu lodi, následně si tudíž zjistíme x-ovou souřadnici lodě a přičteme k ní polovinu šířky lodě krácenou o poloviční šířku střely. Poté je vyhodnocena podmínka, jestli už nemáme „vystřelen“ maximální počet aktivních střel, pokud ne přidáme do seznamu souřadnice nové střely. V případě zasáhnutí nepřítele nebo proletění střely skrz celý displej, vymažeme příslušnou střelu ze seznamu, abychom mohli přidat novou. Zásah nepřítele se stejně jako u lodě kontroluje při posouvání střely.

Jakékoliv posouvání ať už se jedná o nepřátelé, střely nebo mapu je realizováno jednoduchým způsobem. Souřadnice těchto objektů jsou ukládány buďto v poli nebo v případě střel v seznamu. Při každé aktualizaci se tyto hodnoty dekrementují nebo inkrementují v závislosti na typu objektu. Nejběžnější je posun na ose Y, ale sofistikovanější objekty se mohou posouvat i ve vodorovné rovině.

Bitmapy pro nepřátelé jsou při inicializaci uloženy do pole. Při náhrávání souboru s údaji nepřátel jsou načtena i čísla neptřátel. Ty nám pak určí na které souřadnici se má který nepřítel vykreslit. Nepřátelé se vykreslují podle svých pozic uvedených v souboru, při vykreslení všech nepřátel se začnou vykreslovat znova na původních pozicích, proto je třeba tyto pozice uchovávat. Podle druhu nepřítele se určuje jeho chování, jestli bude mít povoleno střílení, jakou bude mít rychlost a další aspekty. Pro uchovávání zasažených nepřátel použijeme pole, kde se bude uchovávat logická hodnota true, pokud nepřítel byl zasažen, jinak false. Po zasažení nepřítele se na jeho místo vykreslí výbuch. Tento výbuch bude znázorněn po dobu 5 cyklů.

Pokud má nepřítel nastavenou střelbu, bude se periodicky vykreslovat střela od nepřítele směrem k lodi, vzdálenost střely je omezená. Rychlost střelby záleží na druhu nepřítele. Po zániku nepřítele zanikne i střela. Střely nepřátel se ukládají kvůli jednoduchému přístupu přímo ve třídě `Nepritel`. Neslabší nepřítel je samozřejmě v prvním kole, tento nepřítel nemá nastavený pohyb do stran, ani povolenou střelbu, jeho rychlost je rovněž nízká.

Detekce kolize s nepřátel s lodí nebo s jejími střelami se ověřuje v odvozené třídě `Kolize`.

```
public boolean overZasah(int x, int y, int vyska, int sirka, boolean skore) {

    for(int i = 0; i < _nepritelX.length; i++)
    {
        if (( _nepritelX[i] <= (x + sirka)
            & (_nepritelX[i] + _nepratele[ _cisloNepritele[i]]. getWidth()) >= x
            & (_nepritelY[i] <= (y + vyska))
            & (_nepritelY[i] + _nepratele[ _cisloNepritele[i]]. getHeight()) >= y
            & _aktivniNepritel[i] )
             ||(( _xStrela[i] <= (x + sirka)
            & (_xStrela[i] + _strelaSirka) >= x
            & (_yStrela[i] <= (y + vyska))
            & (_yStrela[i] + _strelaVyska) >= y)
            & (!skore))
            & _aktivniNepritel[i])
        {
             _aktivniNepritel[i] = false;
            _zasahnutiNepratele[i] = _vybuch.length - 1;
            if (skore)
                _skore += 100 * (1 + _cisloNepritele[i]) ;
            return true;
        }
    }
    return false;
}
```

Výpis 7: Metoda pro detekci kolize mezi nepřítelem a dalším objektem

Detekce kolize se ověřuje při každém pohybu lodi nebo vystřelení a posunu střely. Samotná detekce spočívá v tom, ze se vezmou souřadnice porovnávaného objektu a jeho šířka, případně výška a prochází se jednotlivé souřadnice nepřátel, pokud se zjistí, že se některé souřadnice překrývají vyhodnotí se kolize. V případě zásahu nepřítele naší

střelou bude odebrána střela i nepřítel. Pokud dojde ke kolizi naší lodě a nepřítele, zruší se nepřítel a loď se přesune na počáteční pozici, přičemž bude dočasně neovladatelná. V případě, že jsme dosáhli limitu pro postup do dalšího kola, hra se na chvíli přeruší a načtou se nová data pro další level. To znamená, že se znova provede celá inicializace. Ještě než k tomu dojde si musíme uložit počet životů a dosažené skóre.

5.4 Testování hry

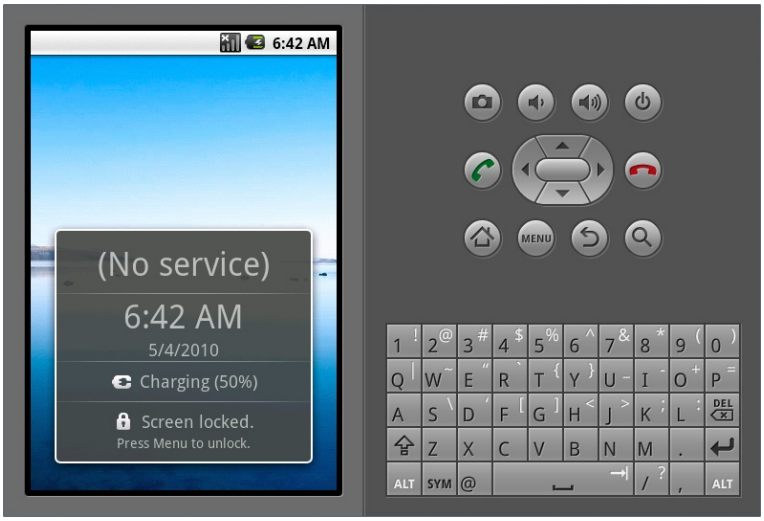
Hra byla testována a optimalizována pouze na Emulátoru obsaženého v Android SDK. Emulátor umožňuje testování a vývoj aplikací, aniž bychom potřebovali skutečné zařízení s OS Android. Tato verze hry je primárně testována na emulátoru používající verzi 1.6, pokud dané fyzické zařízení používá starší verzi OS Android, nebude možno hru na něj nainstalovat, protože není kompatibilní.

Vlastnosti Emulátoru se dají nastavit pomocí nástroje AVD manager, jež nám pomáhá spravovat všechny vytvořené virtuální zařízení. Pomocí tohoto nastavení lze u emulátoru docílít téměř identického chování jaké by mělo mít cílové zařízení. Umožňuje nám nastavit vlastnosti paměťové karty, podporu kamery, rozlišení displeje a spoustu dalších věcí. Pokud již máme vytvořený emulátor podle našich požadavků, můžeme na něj nainstalovat naši aplikaci. Ve vývojovém prostředí Eclipse to uděláme následovně:

1. Spustíme nástroj Run Configurations,
2. přídáme novou Android aplikaci,
3. nastavíme její jméno,
4. vybereme projekt, který chceme spouštět a aktivitu pro spuštění,
5. nakonec nastavíme na které virtuální zařízení se má aplikace naistalovat a spustíme.

V případě úplně prvního spuštění emulátoru, je třeba počítat s časovou prodlevou, která se může zdát dlouhá a emulátor vypadá, že nereaguje. Podle výkonnosti stroje na kterém je emulátor spouštěn se prodleva může pohybovat mezi 5-15 minutami. Při dalším spouštění už bude časová prodleva znatelně kratší.

Při testování jsem narazil na problém při pokusu spustit debug režim, pokud jsem totiž vypnul daný emulátor a po určitém čase ho chtěl opět spustit v režimu pro debugování, tak se emulátor sice spustil, ale nebyl propojený s vývojovým prostředím, v mnoha případech ani nebyla nainstalovaná nová verze aplikace na zařízení, což ztěžovalo a pozdržovalo vývoj asplikace. Tento problém jsem ale vyřešil jednoduchým způsobem, místo zavírání a znovu spouštění emulátoru, jsem tento pouze „odpíchnul“ od vývojového prostředí a při každé nové verzi se opět na něj připojoval. Tento způsob výrazně urychlí a zjednoduší testování dané aplikace. Po spuštění bude emulátor vypadat podobně jako na následujícím obrázku.



Obrázek 11: Ukázka emulátoru verze 1.6 po spuštění

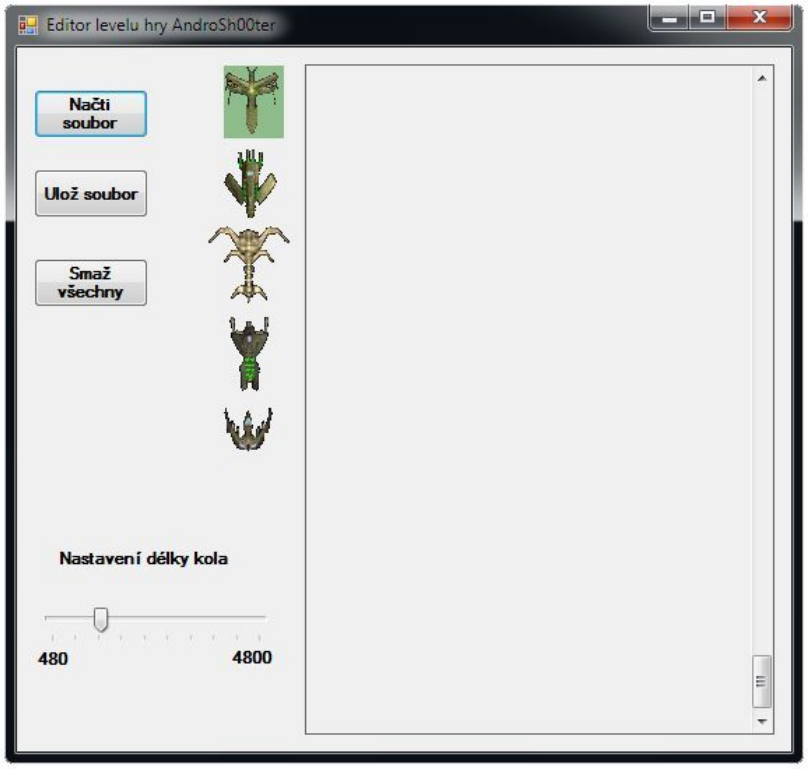
Emulátor má velmi podobné rozložení kláves jako klasické zařízení. Pro odblokování tlačítek je třeba stisknout MENU, poté už můžeme s emulátorem začít normálně pracovat.

Při popisu jádra hry bylo uvedeno, že nastavení jsou načítány z různých souborů. Způsob načítání byl uveden výše, menší problém nástává ale při vkládání načítaných souborů do Emulátoru. K tomuto účelu máme k dispozici celkem solidně vybavený nástroj DDMS, pomocí něhož můžeme lépe optimalizovat a vytvářet aplikace. Najdeme zde všechny spuštěné emulátory, které jsou připojeny k Eclipsu, u každého emulátoru můžeme vidět jeho spuštěné procesy, spuštěná vlákna a další. Mezi velmi užitečné pomůcky toho nástroje patří výpis logu v LogCat tabulce, jsou zde uvedeny veškeré zprávy emulátoru, jakož i námi definované zprávy například při vyjímkách nebo potvrzení různých aktivit. V neposlední řadě nám DDMS nabízí File Explorer, pomocí něhož můžeme velmi jednoduše nahrávat soubory do emulátor, nebo z emulátoru do PC.

5.5 Editor levelů

Jako součást vytvořené hry je přibalen i jednoduchý editor levelů pro PC. Tento editor je vytvořen pomocí programovacího jazyka C# za použití třídy `Windows.Forms`. Díky tomu se vytvoří klasické Windows okno. Pomocí tohoto editoru můžeme rozmisťovat jednotlivé nepřátelé dle vlastního uvážení. I když tohle není úplně pravda, editor má určitá omezení, tak aby nevznikala nesmyslná kola. Editor je možné použít buď pro vytvoření nového levelu, nebo samozřejmě ke změně již vytvořeného levelu.

Jedním z omezení Editoru, je nastavení šířky a délky levelu, jelikož je hra koncipovaná pro vertikální použití s rozlišením HVGA(320x480) je šířka upravovaného levelu pevně nastavena na 320pixelů, výšku je možné nastavovat posuvníkem v rozmezí od 480 do 4800 pixelů, z toho vyplývá, možnost upravit maximálně 10 „obrazovek“. Dalším omezením je počet vkládaných nepřátel, tento počet je nastaven na čtvrtinu výšky upravovaného levelu v pixelech.



Obrázek 12: Editor hry AndroSh00ter

Jak můžeme vidět na obrázku výše, editor je velice jednoduchý a jeho funkce jsou lehce pochopitelné. V levé části editoru se nacházejí tlačítka pro načtení respektive uložení levelu. Další tlačítko umožňuje veškeré vykreslené nepřátele smazat. Vedle tlačítek nalezneme všechny možné nepřátele, kliknutím na některého z nich se vybraný nepřítel označí, toto je signalizováno jinou barvou pozadí. V levé spodní části se nachází posuvník pro určení délky upravovaného levelu. Při každé změně posuvníku budou smazány všichni nepřátelé umístěni mimo nové hranice délky levelu. Selekce těchto nepřátel je prováděna pomocí technologie Ling to SQL.

Poslední čast je určena pro samotnou editaci. Po spuštění bude editor vypadat, přesně jako na obrázku, to znamená, že editační pole bude šedé, pro začátek editace je třeba kliknout do tohoto pole, pro jeho aktivaci. Při dalším stisku se již budou přidávat další nepřátelé a vykreslovat na určená místa. Nepřátelé jsou ukládání do seznamů, které udržují jejich pozici a druh nepřítele.

Při vložení nového nepřítele se kontroluje, jestli se nepokoušíme vložit nepřítele mimo rozměry displeje, v tomto případě budeme upozorněni informativní zprávou. Pokud bychom chtěli vložit nového nepřítele přímo přes nějakého dříve vloženého, bude místo vložení nového, původní nepřítel vyjmut a vkládání se přepne na druh vyjmutého nepřítele. Posuvník v editačním okně nám umožňuje posouvat vykreslovanou plochu v rámci jejích mezí. Při každém posunu se nepřátelé překreslí podle uložených bodů. V následující ukázce můžeme vidět detekci kolize nepřátel pomocí Linq to SQL.

```
var vysledek = (from Point bod in body
                where (x + sirka/2 > bod.X & x < bod.X + sirka/2 & y + vyska/2 > bod.
                    Y & y < bod.Y + vyska/2)
                select bod).FirstOrDefault() ;
return vysledek;
```

Výpis 8: Zjištění kolize přidavaného nepřítele s dříve přidaným

Do editoru si můžeme nahrát už vytvořený level pomocí tlačítka **Načti soubor**, po stisknutí se otevře dialog pro procházení souborů na disku a můžeme si vybrat, který soubor chceme editovat. Tento soubor musí být pojmenován „názevLevelu.dat“. Stejné podmínky platí i při ukládání soubor, kdy nám dialog ani nedovoluje jinou příponu. Pokud bychom ukládaný soubor pojmenovali jinak, při nahrání na zařízení by nedošlo k jeho načtení. Soubor je třeba nahrát do složky ze které se načítají levely, jiná složka není přípustná.

6 Možnosti rozšíření

Navržené a implementované herní jádro hry AndroSh00ter využívá jednoduché herní konstrukce a může sloužit pro mnoho dalších rožšíření. Může být použito jako základ pro další jednoduché hry. Jednou z nejednodušších možností rozšíření je změna lodi vlastní nebo lodi nepřátel, stačí nahrát místo původních vlastní sprity lodí. Další možností je například změna systému ukládání hry, kdy se nebude ukládat pouze jediná hra, ale bude možno nahrát i uložit několik her.

Pro širší možnost použití by mohla být hra předělána pro hru na šířku, případně další pro jiná rozlišení displejů. Taky je možnost dodělat propracovanější dotykové ovládání, které je zde použito pouze ve zjednodušené formě.

7 Závěr

Cílem této práce bylo navrhnout jednoduché herní jádro a jednoduchou hru postavenou na tomto jádře. Hry pro mobilní telefony jsou v dnešní době velice oblíbené a na trhu snad nenajdeme telefon, který by nějakou podobu mobilních her nepodporoval. Velkou výhodou těchto her je, že je můžeme hrát prakticky kdekoliv. Se zvyšujícím se výkonem mobilních telefonů narůstá také kvalita a propracovanost mobilních her. Tyto hry už se od klasických PC her moc neliší, je třeba počítat s menšími rozměry displeje, ale to v mnoha ohledech nehraje roli.

AndroSh00ter je vůbec mou první zkušeností co se týče vývoje her. Platforma Android OS byla pro tento účel zvolena záměrně, jelikož ještě stále patří mezi nováčky na poli mobilních operačních systémů. I díky tomu, že je Android v současnosti vyvíjen pod hlavičkou Google se jeví jako velký příslib do budoucnosti a dle mého názoru může brzy konkurovat takovým gigantům na trhu mobilních telefonů a zařízení jako je Nokia nebo iPhone. Android svá zařízení koncipuje především pro snadné využití Google internetových služeb. Protože Android vychází z Javy, implementované jádro bude po menších úpravách možno použít i pro telefony podporující Javu.

8 Reference

[1] http://www.pocketgamer.co.uk
Novinky ze světa mobilních her

[2] http://www.android.com
Oficiální stránka OS Android

[3] http://www.root.cz
Informace nejen ze světa Linuxu

[4] http://softsters.wordpress.com
Blog o software

[5] http://cs.wikipedia.org
Otevřená encyklopedie

[6] www.droidnova.com
Blog o vývoji Android aplikací

[7] Silva V.,
Pro Android Games, Apress, 2009

A Návod ke hře

Pro ovládání hry je primárně určen D-Pad, případně klávesnice, lze použít i zjednodušené dotykové ovládání:

- směr nahoru: tlačítko „UP“ na D-padu, případně tlačítko „O“, nebo kliknutí na loď a potažení nahoru
- směr dolů: tlačítko „DOWN“ na D-padu, případně tlačítko „L“, nebo kliknutí na loď a potažení dolů
- směr doprava: tlačítko „RIGHT“ na D-padu, případně tlačítko „W“, nebo kliknutí na loď a potažení doprava
- směr doleva: tlačítko „LEFT“ na D-padu, případně tlačítko „Q“, nebo kliknutí na loď a potažení doleva
- vyslání střely: tlačítko „CENTER“ na D-padu, případně tlačítko „mezerník“, nebo kliknutí na loď
- pauza: klávesa „P“
- konec hry: tlačítko „esc“ nebo „zpět“

Ve hře se nachází 5 nepřátel, uvedených na další straně. Podle pořadí jsou ohodnoceni 100 - 500b za zásah.

Obrázek 1: Ukázka nepřítele 1

Obrázek 2: Ukázka nepřítele 2

Obrázek 3: Ukázka nepřítele 3

Obrázek 4: Ukázka nepřítele 4

Obrázek 5: Ukázka nepřítele 5